FINAL

Integrating Model-based Artificial Intelligence Planning with Procedural Elaboration for Onboard Spacecraft Autonomy

Russell Knight, Steve Chien, Erann Gat, Tom Starbird, Kim Gostelow, Bob Keller, Weldon Smith

Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive, M/S 126-347, Pasadena, CA 91109-8099
email: {firstname.lastname}@jpl.nasa.gov

Abstract

This paper describes the integration of a model-based planner into a procedural architecture. This architecture is unusual in that the internal procedures and the planners inter-operate in an asynchronous fashion in a multi-threaded environment via a goal-based interface. The submission of goals may trigger either internal procedures or planners, and both internal procedures and planners may submit goals. The procedural architecture described is the Mission Data System (MDS) Goal Achieving Module (GAM) architecture. The planner described is the CASPER (Continuous Activity Scheduling Planning Execution and Replanning) system. This approach has been prototyped and tested against virtual spacecraft and comet-lander operations scenarios and simulations.

Introduction

The MDS (Mission Data System) is a multi-mission combined flight and ground software designed to enable significant autonomy in spacecraft operations. A definition of a spacecraft/scenario consists of modules that are responsible for maintaining the state of the spacecraft and achieving high level goals while reacting to unforeseen changes onboard and in the environment.

CASPER (Continuous Activity Scheduling Planning Execution and Replanning) is a soft real-time planning system that achieves high level goals while monitoring the state of the spacecraft and environment, as well. The definition of a spacecraft/scenario for CASPER consists of a declarative model consisting of activities, states, resources, and their relationships.

The difference between these systems is that the modules that make up a spacecraft/scenario definition in the MDS are procedural—that is to say, they are (somewhat) arbitrary pieces of code. The definition of a spacecraft/scenario for CASPER is (mostly) declarative. Both have advantages and disadvantages. Arbitrary code can be very specialized and fast, but is difficult to validate and difficult to produce. Validation tools exist for declarative models, and the development of these models is fairly straightforward, but generic algorithms for reasoning about all possible domains do not yet exist (and in our opinion are not likely to any time soon). We present a mapping from the requirements of MDS modules to the capabilities of CASPER that enables a mixed mode of procedural/declarative modeling while providing monitoring, prediction, and validation of state.

Overview

We first describe the goal-based interface. This is the glue that holds the MDS together. Then we describe procedural units called GAMs. We follow with a description of CASPER and its associated activity-execution/state-update interface. We describe a mapping to and from both interfaces. Finally, we present results demonstrating the feasibility of this approach.

The Goal-Based Interface

A goal is a constraint on a state-variable over time.

A state-variable is the representation of state used to reason about actual states of the spacecraft, e.g. propellant level, orientation, or memory usage. A state-variable may also represent an abstraction such as the number of high quality pictures stored in memory. A constraint on a state-variable is an expression of what values the state-variable may be and still satisfy the goal, e.g. orientation must be pointing at Europa implies that, of all values for orientation possible, only those where the spacecraft (or the camera) is pointing at Europa are valid.

Goals constrain state-variables over a specified time or interval. This interval is expressed as two ordered time-points—a start-time and an end-time.

At any point in time throughout its life, a goal finds itself in one of many states referred to as its*status*. The status consists of an *outcome*, *stability*, and *time*. Outcome describes whether the goal is achieved or failed (or predicted to be achieved or failed.) Stability indicates whether or not the system is performing computations that might result in a change in the outcome (e.g. during elaboration), and time indicates whether the goal is pending, currently active, or past.

The interface upon which the architecture is based is quite simple. State-variables receive goals and route these to a dedicated module that is responsible for status updates to every goal submitted to it. It is also responsible for submitting new goals to ensure that the current goals under its purview are achieved. These modules are Goal Achieving Modules or GAMs (described more fully later.) Thus, each GAM must provide the facility for accepting submitted goals. In a sense, we can see each GAM as a node connected to the goal-cloud via state-variables, much as computers connected to a network. Each node is neither concerned with how the information is passed nor with how it was generated, only that it must be fulfilled. Any GAM can submit goals to the cloud, and any GAM might receive goals from the cloud (see Figure 1).

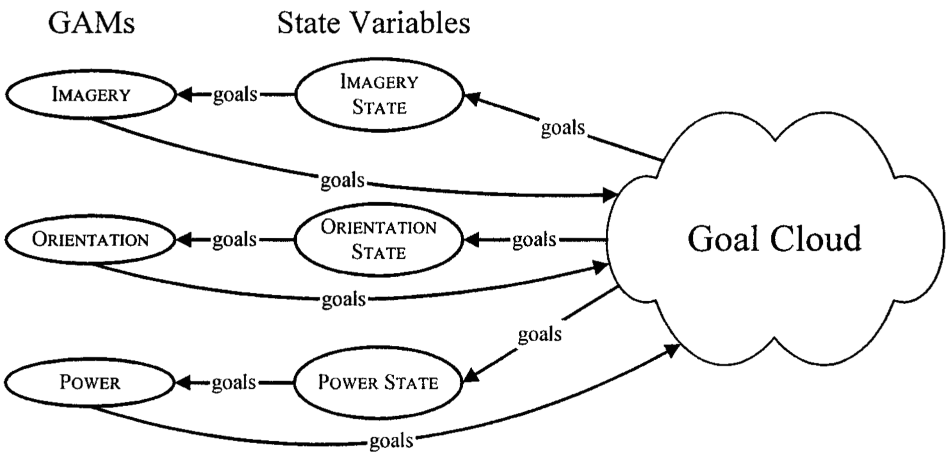

Figure 1. The Goal Interface as a Goal Network Cloud

The Goal Achieving Module

The basic function of the GAM is to maintain constraints on state-variables over time. These constraints are received through the goal-based interface as goals. Each state-variable in the system has one associated GAM. A single GAM may be responsible for many state variables but must be responsible for at least one (otherwise, it cannot receive goals.)

Upon receipt of a goal, a GAM must decide if this new goal can be achieved. The GAM must decide on one or more of the following:

- the goal is already satisfied (enough power is available during that time)
- rescind other conflicting goals to accommodate the new goal
- perform actions that will accommodate the new goal
- sub-goal to other GAMs in hopes of accommodating the new goal.

This decision process is known as *elaboration*. The result of elaboration is a guarantee that the goal will be achieved over the period specified assuming the predicted future state is within tolerance of the actual future state.

For example, an ORIENTATION GAM (or simply ORIENTATION) may receive a request from the IMAGERY GAM (or simply IMAGERY) in the form of a pointing goal. (Note: since state variables act as routers for goals and do not actually change them in any way, we will henceforth talk about goals being received directly by GAMs with the understanding that the goals are actually routed through the state-variable constrained by the goal.) In general, a goal describes a time-window that the goal must be true: in our example, the orientation of the spacecraft must be consistent with the pointing request. ORIENTATION might be capable of achieving this without further sub-goaling. But, ORIENTATION may decide on a course of action that includes sub-goaling, such as using reaction wheels to achieve the pointing goal. Reaction wheels require the use of power, so a power goal would be issued to POWER, and so on, until all goals can be achieved (see Figure 2). Of course, it may be the case that a goal is unachievable. If so, the goal is reported as such by the receiving GAM, and the issuing GAM must decide on what operations make sense to achieve its outstanding goals in light of this unachievable sub-goal.

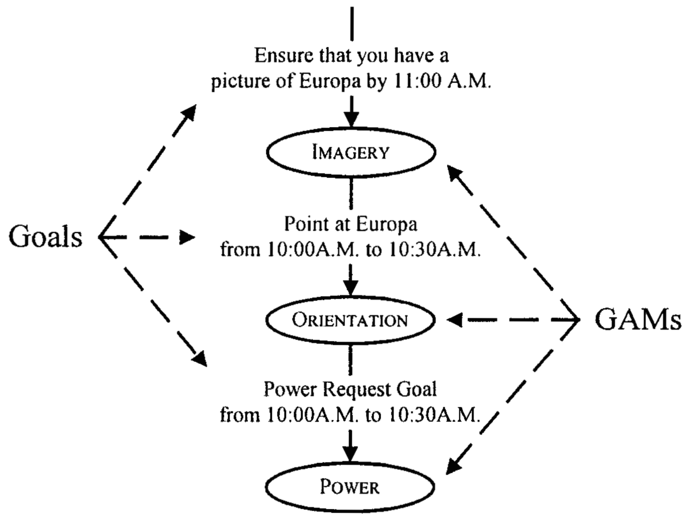
Ensure that you have a picture of Europa by 11:00 A.M.
IMAGERY
Goals
Point at Europa from 10:00A.M. to 10:30A.M.
ORIENTATION
GAMs
Power Request Goal from 10:00A.M. to 10:30A.M.
POWER

Figure 2. Elaboration Example

Elaboration is only one facet of achieving goals; spacecraft operations demand execution and monitoring, as well. Through a system checking preconditions of the goal, the lower level controllers are notified as to the state they must maintain at any given time based on the time-points. Therefore, execution is simply the commanding of modes to lower level controllers.

Monitoring is the process of updating state estimates based on measurements. Measurements result from a process of combining appropriate information and reporting these to a measurement model. It is up to the measurement model to update the state appropriately.

The overall relationship between the modules described is represented in Figure 3.

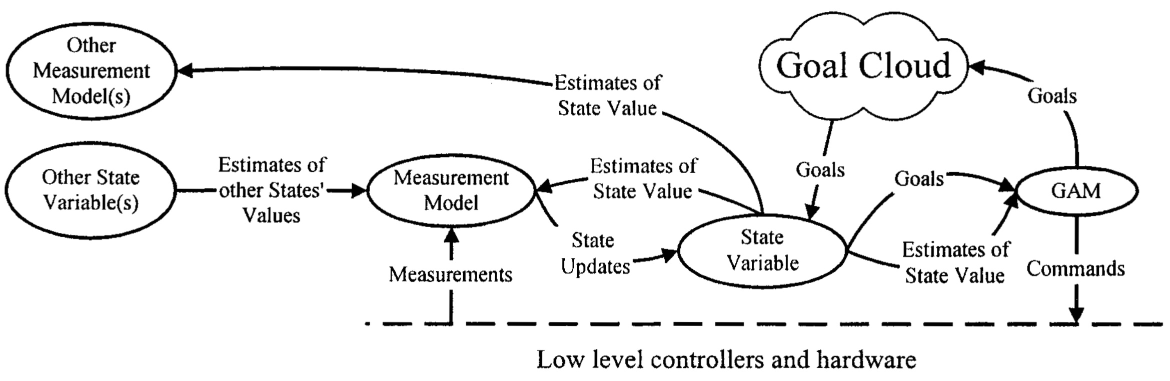

Figure 3. MDS Modules and Interfaces

Note that the interfaces are simple: State Variables receive updates and provide estimated values; GAMs receive goals and estimated values and provide goals and commands; Measurement models receive measurements and estimated values and provide state updates.

CASPER

Spacecraft operations demand a high level of responsiveness in dynamic environments. To provide the required level of responsiveness while including a declarative modeling environment, we utilize a *continuous planning* approach and have implemented a system called CASPER.

Traditionally, declarative planners provide execution capabilities based on a batch formulation of the problem. In the batch approach, time is divided up into a number of planning horizons, each of which lasts for a significant period of time (see Figure 4). When one nears the end of the current horizon, one projects what the state will be at the end of the execution of the current plan. The planner is invoked with a new set of goals and this state as the initial state (for example the Remote Agent Experiment operated in this fashion [Pell et al, 1997]).

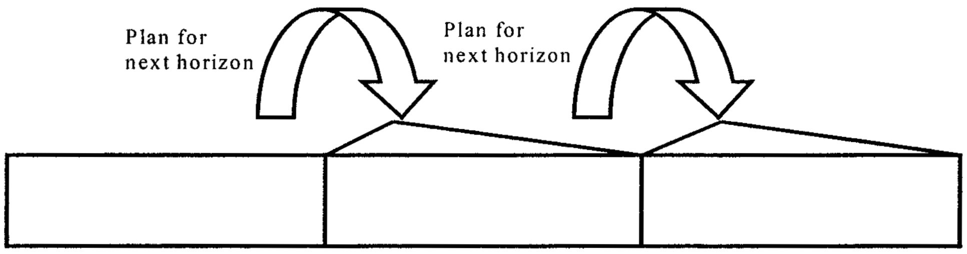

Figure 4. Traditional Batch "Plan the Execute" Cycle.

This approach has a number of drawbacks. In this batch oriented mode, typically planning is considered an off-line process which requires considerable computational effort and there is a significant delay from the time the planner is invoked to the time that the planner produces a new plan.[1] If a negative event occurs (e.g., a plan failure), the response time until a new plan may be significant. During this period the system being controlled must be operated appropriately without planner guidance. If a positive event occurs (e.g., activities finishing early), again the response time may be significant. If the opportunity is short lived, the system must be able to take advantage of such opportunities without a new plan (because of the delay in generating a new plan). Finally, because the planning process may need to be initiated significantly before the end of the current planning horizon,

[1] As a data point, the planner for the Remote Agent Experiment (RAX) flying on-board the New Millennium Deep Space One mission (Muscettola et al 1997) takes approximately 4 hours to produce a 3 day operations plan. RAX is running on a 25 MHz RAD 6000 flight processor and uses roughly 25% of the CPU processing power. While this is a significant improvement over waiting for ground intervention, making the planning process even more responsive (e.g., on a time scale of seconds or tens of seconds) to changes in the operations context, would increase the overall time for which the spacecraft has a consistent plan.

it may be difficult to project what the state will be when the current plan execution is complete. If the projection is wrong the plan may have difficulty.

Rather than considering planning a batch process in which a planner is presented with goals and an initial state, the planner has a current goal set, a plan, a current state, and a model of the expected future state. At any time an incremental update to the goals, current state, or planning horizon (at much smaller time increments than batch planning)[2] may update the current state of the plan and thereby invoke the planner process. This update may be an unexpected event, the receipt of a new goal, or simply time progressing forward. The planner is then responsible for maintaining a consistent, satisficing plan with the most current information (see Figure 5). This current plan and projection is the planner's estimation as to what it expects to happen in the world if things go as expected. However, since things rarely go exactly as expected, the planner stands ready to continually modify the plan.

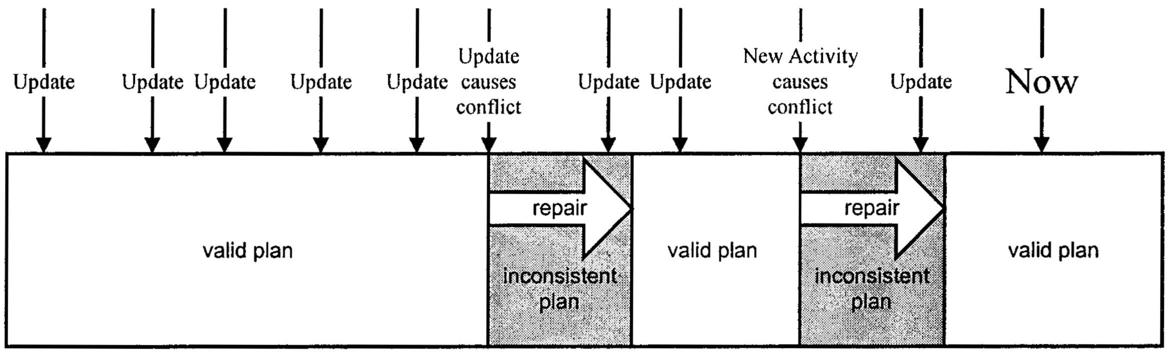

Figure 5. Plan Repair in Response to Updates

As illustrated in Figure 5, CASPER receives state updates. CASPER also executes commands based on activities currently planned and scheduled. Figure 6 shows the interfaces in CASPER.

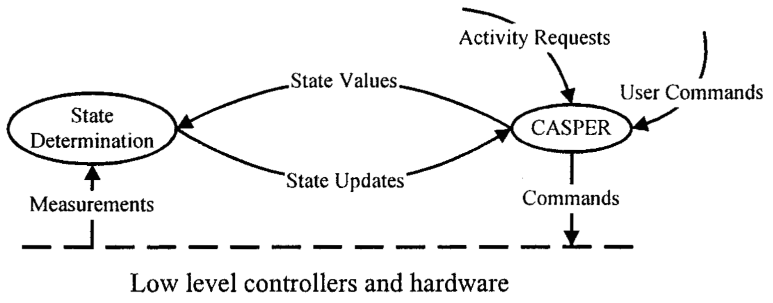

Figure 6. CASPER Module Interfaces

State Determination performs a function similar to measurement models in MDS in that it takes as input the current estimated state(s) and outputs state updates. CASPER itself receives as input activity requests, user commands and state updates, and it outputs commands and state values.

A Mapping from Goal-based Interface to the Activity-Execution/State-Update Interface

To map CASPER into the MDS, we must first define which modules CASPER replaces and define a mapping from the interfaces for these modules to the interfaces provided by CASPER. Conceptually, CASPER acts as a stand in for Stave Variables and GAMs. A key notion of this integration is that the machinery or planning (whether performed by an AI planner or by code within a GAM/State Variable) is hidden from an external goal-submitting agent. The interface presented is identical. The dashed box

[2] For the spacecraft control domain we envision an update rate on the order of 10 seconds real time.

in Figure 7 delineates the modules to be swapped out. This box indicates the meta-module that CASPER functions as.

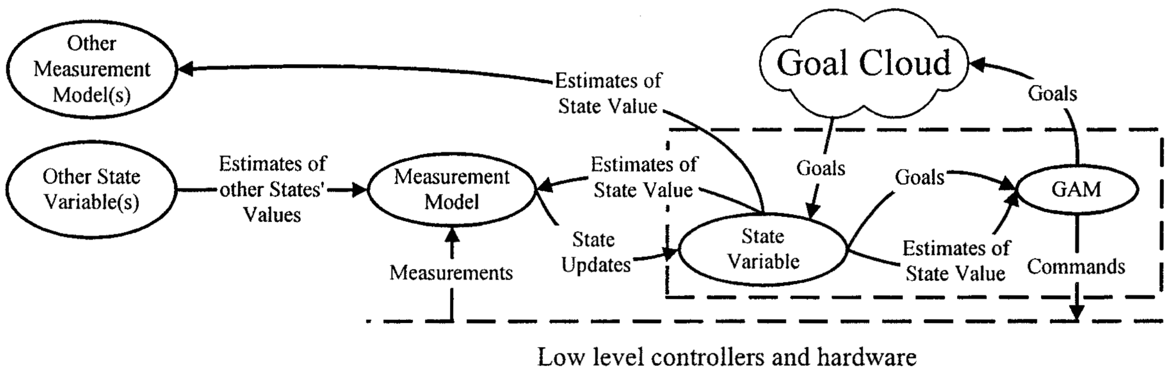

Figure 7. State Variable and GAM modules to be replaced by the Meta Module

This leads to the following simplified meta-module (see Figure 8.)

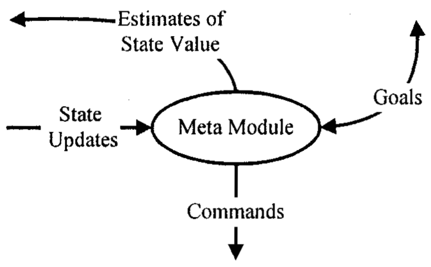

Figure 8. The Meta Module standing in for a GAM and its associated State Variable

As illustrated in Figure 6, CASPER already provides an interface for state updates, estimates of state value, and commands, but lacks a clear interface with respect to goals, both in terms of receiving goals and in terms of creating new goals. CASPER can accept new activities, so we need to provide a mapping between activities and goals.

An activity is similar to a goal in that it is partially defined by an interval that consists of a start- and end-time. But, an activity need not represent a constraint on a state at all. In fact, many activities simply represent an abstraction that is later "fleshed-out" in more detail according to decomposition rules in the model. So, the definition of an activity is less constrained in theory than that of a goal. But, an activity may have constraints on shared states and resources. These are roughly synonymous with state variables. These constraints usually span the temporal extent of the activity, and therefore are a very close match to a goal. Thus, we can define special activities that represent the same semantics as goals. This provides us the interface to receive goals (by converting them to appropriate activities.)

However, we still require an interface that allows CASPER to submit goals that constrain external state variables. Again, we can create an activity, and upon creation of these activities we can convert them to goals and submit them. But, how do we keep track of the status of the activity? What if the receiving GAM modifies the goal by constraining the times when it can be executed? We still require a communication path that informs CASPER of when a sub-goal is achievable.

When CASPER requires an activity, it adds it to the plan and schedules it. Normally, CASPER knows all about the activity, so it can reason about it without external assistance. Depending on the permissions of the activity, CASPER may then attempt to move it or delete it or perhaps change the value of one of its parameters. The model dictates control of each of these operations with permissions e.g. if an activity does not have "move" permission, then CASPER cannot move it.

One of the myriad operations CASPER performs is scheduling an activity. Scheduling an activity is the act of constraining states and resources according to the model of the activity. If CASPER does not have "schedule" permissions for an activity, it can never satisfy the constraints of the activity because it will never be allowed to do so. But, if some external agent (a user, perhaps) overrides the permissions and schedules the activity, then CASPER can continue. This is the mechanism through which we allow the submission goals to external agents. First, we detect and report activities that do not have "schedule" permission to an activity-to-goal translator. This translator then creates a goal and submits it to the goal-cloud while monitoring the goal's status. When the status reports back that it will be achieved, the unscheduled activity is scheduled by overriding the permissions via the user command channel, thereby closing the loop.

Also, CASPER must keep a skeleton model of all state variables that it wishes to constrain via sub-goals submitted to GAMs. Note that this is synonymous with bestowing to GAMs the knowledge of how to express goals.

Our final architecture with the new modules that performs the same functions as the meta module of Figure 8 is shown in Figure 9. Note that the unscheduled activity reporter is imbedded directly in CASPER due to the special knowledge requirements demanded by its function.

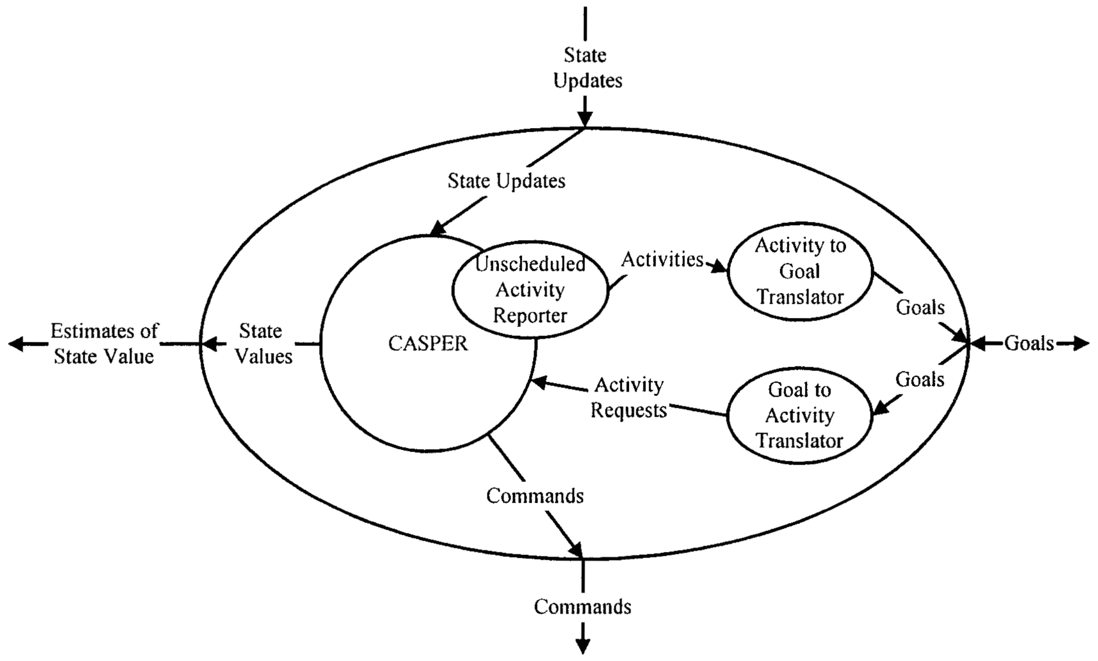

Figure 9. CASPER and supporting modules as the Meta Module

Prototypical Results

We have demonstrated this approach in two domains. The first is the VSS (Virtual Spacecraft Scenario)—a synthetic domain designed to represent a simple spacecraft. The second is the ST4 (Space Technologies 4/Champollion) scenario—a comet-lander drilling, testing samples, observing surface features and effects, and reporting results to an orbiting spacecraft. Our approach worked well in both scenarios in that the CASPER meta module performed at least as well as the procedural modules.

The Virtual Spacecraft Scenario

The purpose behind a simple scenario is to test ideas without the overhead of high fidelity simulators. Toward this end, the VSS was developed. It consists of a spacecraft with one axis of control (much like a phonograph turntable, for those who remember such things), a camera, solar panels mounted on a gimbal with limited motion, a power bus, and batteries.

An example of a high level goal that would be sent would be "ensure that a picture of object X is available at 5:00 P.M." This is a constraint on the SCIENCE state variable. A simplified example of elaboration of this goal would proceed as shown in Figure 10 (nodes are state variable/GAMs and arcs are goals). The numbering indicates a possible ordering of decisions by the system even though elaboration is a potentially parallel process.

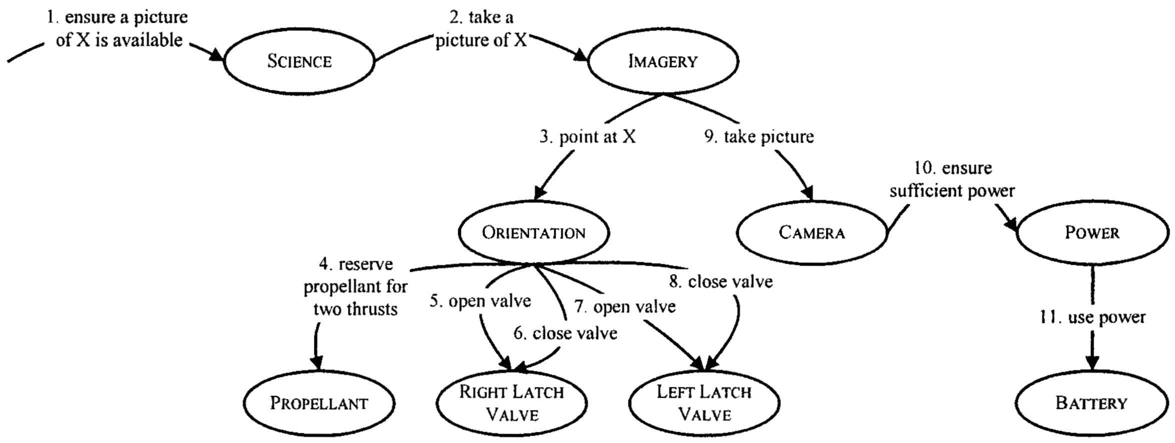

Figure 10. Elaboration of goal in VSS

For simplicity we do not indicate times for the goals, but a temporal ordering is imposed by the system to ensure proper execution.

Consider the operation performed by ORIENTATION in response to the "point at X" goal. When constructing a GAM, information about turning and propellant usage is encoded in implementation code. As the GAM is implemented using standard coding techniques, it is difficult to generate, verify, and maintain. CASPER allows the same behavior while employing a declarative representation, thereby cutting development time while enabling proven verification techniques to be applied to the model. Maintenance time is also cut in that the encoding for the domain is more readable and changes are easier to implement.

Each time a pointing goal appeared in the goal cloud, the goal-to-activity translator would create an appropriate pointing_activity. The elaboration of this activity is encoded in its definition (as above.) It is possible that we would oversubscribe a resource, e.g. using "10" of propellant may oversubscribe our existing store. In this case, CASPER uses its generic algorithms to plan around the oversubscription, and solve it through any number of means.

We first modeled this scenario entirely using GAMs and state variables. Once this operated properly, we implemented pieces in CASPER, until we had example of system performance with varying levels of GAMs versus CASPER elaboration and state representation. As was expected, we found that procedural GAMs took longer to encode than their declarative counter parts. We also found that performance was comparable between the two, both in terms of computation time and in terms of solution quality.

Space Technology 4/Champollion Comet Lander

This scenario entails the landed operations of a comet lander. The lander must drill core samples, test them in an oven, and report information via a communications link. Also, it must take photos of features on the surface and react to interesting events such as the out-gassing of material.

A major difference in terms of our tests between ST4 and VSS is that we have a high fidelity simulator for ST4. Even though we had the demands of a more challenging scenario, we found that most state variables could be represented within CASPER, and similarly, we found that the development time for models expressed in a declarative manner was shorter than those expressed procedurally.

Conclusions

We have described the integration of CASPER, a model-based planner, into the MDS GAM framework, a procedural architecture. This integration was facilitated by the fact that the internal procedures and the planners inter-operate via a goal-based interface. We demonstrated our approach by prototyping and testing against VSS and ST4. We have observed that declarative modeling reduces development time while still providing solutions to rich domains.

Acknowledgements

This work was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. Portions of this work were supported by: the Autonomy Technology Program, managed by Dr. Richard Doyle and with Melvin Montemerlo as the headquarters program executive, NASA Code SM and by the Mission Data Systems Project, managed by Allan Sacks, NASA Code S, and by the JPL Telecommunications and Mission Operations Technology Program, Mission Services Element managed by Peter Shames.

References

P. Backes, G. Rabideau, K. Tso, S. Chien, "Automated Planning and Scheduling for Planetary Rover Distributed Operations," *Proceedings of the IEEE Conference on Robotics and Automation,* Detroit, Michigan, May 1999.

S. Chien, N. Muscettola, K. Rajan, B. Smith, G. Rabideau, "Automated Planning and Scheduling for Goal-based Autonomous Spacecraft," *IEEE Intelligent Systems*, September/October 1998, pp. 50-55.

S. Chien, G. Rabideau, J. Willis, T. Mann, "Automating Planning and Scheduling of Shuttle Payload Operations," Artificial Intelligence Journal 114 (1999) 239-255.

S. Chien, A. Barrett, T. Estlin, and G. Rabideau, "A Comparison of Coordinated Planning Methods for Cooperating Rovers," Fourth International Conference on Autonomous Agents, Barcelona, Spain, June 2000.

S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau, "Using Iterative Repair to Improve Responsiveness of Planning and Scheduling," Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling, Breckenridge, CO, April 2000.

S. Chien, R. Knight, T. Starbird, K. Gostelow, R. Keller, W. Smith, et al. , "Integrating Model-based Artificial Intelligence Planning with Procedural Elaboration for Onboard Spacecraft Autonomy, "SpaceOps 2000, Toulouse, France

B. Engelhardt, S. Chien, D. Mutz, "Evaluating Hypothesis Generation Strategies for Adaptive Problem Solving", Proceedings of the IEEE Aerospace Conference, Big Sky, MT, March 2000.

T. Estlin, T. Mann, A. Gray, G. Rabideau, R. Castano, S. Chien and E. Mjolsness, "An Integrated System for Multi-Rover Scientific Exploration," Sixteenth National Conference of Artificial Intelligence, Orlando, FL, July 1999.

F. Fisher, T. Estlin, D. Mutz, S. Chien, "Using Artificial Intelligence Planning to Generate Antenna Tracking Plans", Eleventh Annual Conference on Innovative Applications of Artificial Intelligence, Orlando, Florida, July 1999.

F. Fisher, R. Knight, B. Engelhardt, S. Chien, N. Alejandre, "A Planning Approach to Monitor and Control for Deep Space Communications", Proceedings of the IEEE Aerospace Conference, Big Sky, MT, March 2000.

A. Jonsson, P. Morris, N. Muscettola, K. Rajan, and B. Smith, "Planning in Interplanetary Space: Theory and Practice" in Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, CO, April 2000

G. Rabideau, R. Knight, S. Chien, A. Fukunaga, A. Govindjee, "Iterative Repair Planning for Spacecraft Operations in the ASPEN System," International Symposium on Artificial Intelligence Robotics and Automation in Space, Noordwijk, The Netherlands, June 1999.

G. Rabideau, B. Engelhardt, S. Chien, "Using Generic Preferences to Incrementally Improve Plan Quality," in Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling, Breckenridge, CO. April, 2000.

R. Sherwood, A. Govindjee, D. Yan, G. Rabideau, S. Chien, A. Fukunaga, "Using ASPEN to Automate EO-1 Activity Planning," *Proceedings of the 1998 IEEE Aerospace Conference*, Aspen, CO, March 1998.

R. Sherwood, T. Estlin, S. Chien, G. Rabideau, B. Engelhardt, A. Mishkin, B. Cooper , "An Automated Rover Command Generation Prototype for the Mars 2001 Marie Curie Rover," SpaceOps 2000, Toulouse, France, June 2000.

B. Smith, B. Engelhardt, R. Knight, D. Mutz, "Automated Planning for Spacecraft and MissionDesign ," Third International Symposium on Intelligent Automation and Control, Wailea, Hawaii, June 2000.

B. Smith, B. Engelhardt, D. Mutz, "Automated Planning for the Antarctic Mapping Mission," Working Notes of the 2nd International Workshop on Planning and Scheduling for Space," San Francisco, CA, March 2000, pp. 184-186.

J. Willis, G. Rabideau, C. Wilklow, "The Citizen Explorer Scheduling System, "Proceedings of the IEEE Aerospace Conference, Aspen, CO, March 1999.